RSRE
MEMORANDUM No. 4215

ROYAL SIGNALS & RADAR ESTABLISHMENT

NON-SEPARATE ARITHMETIC CODES

Author: I K Proudler

ROYAL SIGNALS AND RADAR ESTABLISHMENT

Memorandum 4215

TITLE: NON-SEPARATE ARITHMETIC CODES

AUTHOR: I K Proudler

DATE: September 1988

Squared

SUMMARY

It is shown that a non-separate arithmetic code that preserves both addition and multiplication must be an AN code where the generator A is an idempotent element of the ring being used. An idempotent element is one that satisfies the equation $x^2 = x$. Given this type of code, its ability to detect errors in arithmetic expressions is explored and shown to be poor, due to error masking in multipliers.

The constraints placed on a non-separate multiplication-preserving arithmetic code that avoids such problems are discussed. The simplest code satisfying these conditions turns out to be an AN+B code where both A and B are idempotent elements. Conditions for the existence of this type of code are given along with a list of examples. The fault tolerance provided by these codes is then considered for a specific example.

CONTENTS

1. INTRODUCTION

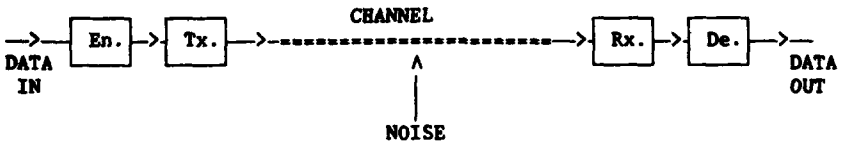
There has been a lot of interest, in recent years, in the design of fault-tolerant arithmetic circuits. This is primarily due to various side-effects of the increasing component densities to be found in modern integrated circuits (VLSI, WSI etc.). Ageing effects and transient malfunctions [1] as well as production faults [2][3] can contribute to the failure of a device. The smaller the feature size is, however, the larger the effect of a given fault. Thus high density integrated circuits face the prospect of lover production yields and reduced reliability in service. It is also becoming more difficult to test integrated circuits, as the increase in component density allows much more complex circuits to be fabricated.

Whereas reliability has always been a goal in the design of electronic circuits, there is now also a need to actively design for increased production yields and reduced testing times. If this is not achieved the component cost of VLSI circuits could be prohibitively large. One solution to all three problems is to incorporate redundancy into the circuit and make the circuit fault tolerant (which also implies self-testing). Fault-tolerant arithmetic [4] is one of the many different approaches to the topic of fault-tolerance.

First Diamond [5] and Brown [6] and then others (see [7] p86) investigated the use of error-correcting AN-codes to detect and correct errors. The majority of this work has been directed towards the protection of the addition operation and the theory is extensive [7]. Multiplication can, of course, be thought of as a series of additions and thus, in theory, be protected by the same techniques. This however means that only one of the operands can be in encoded form, for the other operand is used to control this sequence of additions. This latter process is unlikely to commute with the encoding function of an addition preserving code and thus this operand can not be encoded. Hence errors can only be detected in one of the two operands using this method. More significantly, this approach limits the application of AN codes to individual fault tolerant adder or multiplier circuits. The environment of modern integrated circuits (see above) means, however, that such an approach is unlikely to work.

Arithmetic error-correcting codes were initially conceived as a generalisation of the very successful error-correcting data transmission codes [8][9]. There are many parallels between the two types of code ([7] ch.8) but a significant difference is the fact that the encoding and decoding circuitry for arithmetic codes may itself be prone to errors. In the data transmission problem (figure 1) the assumption is that the only point where the data can be corrupted is the communications channel. Thus the encoder and decoder circuitry can be as

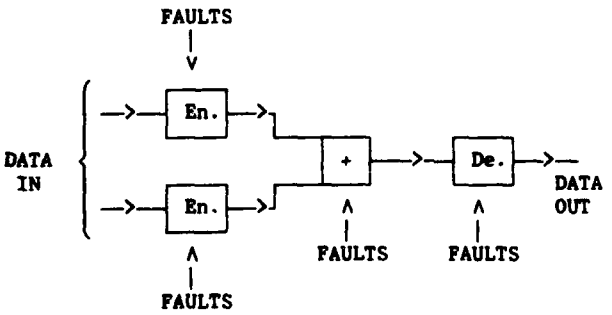
complex as desired, subject only to practical requirements (i.e. component cost, through-put, etc.), and still function correctly.



En. - E-C code encoder
Tx. - channel transmitter
Rx. - channel receiver
De. - E-C code decoder

Figure 1. Error Protected Communication Channel.

In the case of arithmetic circuits (figure 2), the need for error-correction arises because of imperfections in the fabrication of the circuits [2][3] and the various fault-inducing physical effects that occur during the life-time of the device [1]. Thus not only will the arithmetic circuit (the adder in figure 2) be prone to faults but so also will any other piece of circuitry (e.g. the decoder). This, given a uniform distribution of faults throughout the circuitry, creates a special problem. If the error-correcting code is such that the encoder and/or decoder is more complex than the circuit it is supposed to be protecting then it is more likely that a fault will occur in the encoder and/or decoder than in the original circuit.



En. - E-C code encoder
+ - adder
De. - E-C code decoder

Figure 2. Fault Tolerant Arithmetic Circuit.

A lot of the work done to date has concentrated on the error detection/correction properties of the codes and has tended to ignore this rather fundamental problem. One solution is to build fault-tolerant encoders and decoders using the techniques of self-checking circuits [10]. The level of complexity of

such circuits may well be limited[1] by the need to make them self-checking. Another option would be to use a replicating technique like TMR [11] but this can be expensive, in terms of hardware, for complex circuits. An alternative scheme is, perhaps, to ensure that the encoder and decoder circuitry is relatively small compared to the circuit being protected. In which case, although it is still possible for the encoder and/or decoder to become faulty, in all probability it will be the arithmetic circuits that fail first. Abraham [12] has recently introduced just such an error protection scheme, which he calls "algorithm fault-tolerance". To date [13][14][15][16] this type of scheme has only been applied to the protection of systolic arrays that perform matrix manipulations.

In this paper we consider an approach and address the problem of classification of error-detecting codes that can protect large arithmetic expressions involving only addition and multiplication, or more correctly circuits that implement such expressions. In such a system (Figure 3) the inputs to the arithmetic circuit are first encoded, then manipulated in the required manner (multiplications and additions) and finally the outputs are checked for the presence of an error. Such a scheme would be eminently suitable for Digital Signal Processing applications where most of the computation reduces to series of multiplications and additions.

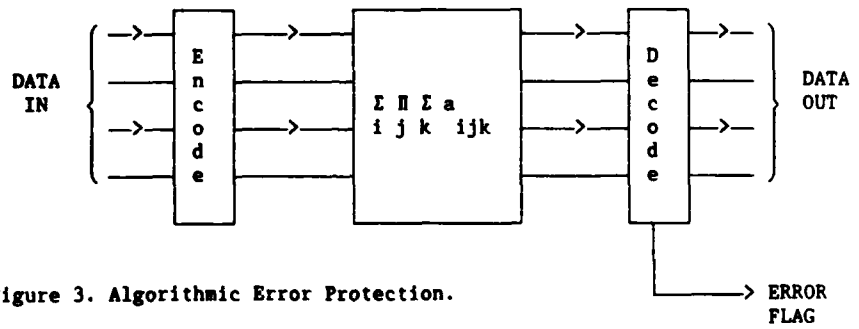


Figure 3. Algorithmic Error Protection.

In order for it to be possible to check this type of arithmetic expression it is necessary for the error-detecting code to preserve the operations of addition and multiplication. It is well known ([7] section 3.3) that if an addition preserving code is a separate code (i.e. with independent information and check

1. A better understanding of the techniques of totally self checking circuits is required before a definitive statement can be made.

parts) then it must be, to within isomorphism, a residue code. Such codes also preserve multiplication. The properties of residue codes are well known [7] and not considered here, except to note that in principle they require a specialised implementation with separate information and check circuitry. This may or may not be a drawback depending on the application. Certainly the need for a specialised architecture precludes the use of separate codes for non-intrusive, real-time testing applications.

In section 2 we derive the form of a non-separate multiplication/addition preserving arithmetic code and show that the error protection ability of such a code is poor due to error masking in multipliers. In section 3 we consider the requirements placed on a non-separate arithmetic error detecting code by the need to avoid error masking in the multiplication process. The types of error to be found in a multiplier are discussed and a set of constraints thus imposed on the code is found. The theoretical properties of the simplest code to satisfy these constraints are studied in section 4 and some examples are given. Using a specific example, the fault tolerance provided by this type of code is considered in section 5 and the results of some computer simulations are presented in section 6.

2. MULTIPLICATION/ADDITION PRESERVING ARITHMETIC CODES

A multiplication/addition preserving arithmetic code is a redundant encoding of a finite set of integers (Z_n say) such that their arithmetic structure is preserved. It is usual [7] to consider the arithmetic to be modulo the integer n ($n=2^b$ for two's complement, $n=2^b-1$ for one's complement) in which case the set Z_n can be considered to be a ring [17]. As the encoded integers are to be manipulated by a computer, or similar dedicated hardware, they too will be elements of some integer ring (Z_m, $m>n$, say). In the appendix (subsection 11.1) we show that this leads to the result that the encoding function, for an integer x, has to be of the form Ax, where A is such that $A^2 = A \text{ MOD } m$ (i.e. an idempotent: see subsection 11.2).

Following standard coding theory [7], we define an "error" to be a non-zero difference between a variable's actual value and the correct one. The term "fault" is used to mean a physical failure which then results in an error, in one or more variables.

The ability of AN codes to detect, and correct, errors in addition is well documented [7], so here we only consider the process of multiplication. Figure 4 shows the situation where the each input to a multiplier consist of a valid codeword and an error (e.g. Ax, e_x respectively). The output contains terms from the multiplication of the two inputs and a term (e_{xy}) that represents any error

in this process.

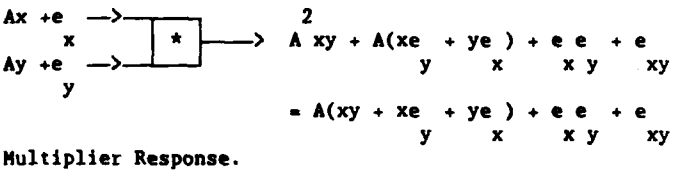


Figure 4. Multiplier Response.

The basic principle of error detection using AN codes is that valid codewords are multiples of the generator (A). An error can be detected if it changes a codeword into a value that is not a multiple of A (not a codeword). Consideration of the multiplier output thus reveals the following facts:

a) faults in the multiplier (e_{xy}) are detectable (provided the associated errors are not multiples of A).

b) if only one input is in error (i.e. one of e_x, e_y is zero) then the error in the other input is masked since it appears as a multiple of A.

c) if both inputs are in error this fact can be detected through the term $e_x e_y$. Error correction will not be possible because the individual errors (e_x, e_y) cannot be recovered from their product.

It is easy to see that the above coding scheme will mask (the effects of) a single fault at the input to a multiplier. Adders do not mask faults so if this multiplier is part of a network consisting of just additions and multiplications then any single fault occurring before the last multiplier will be masked. Only single faults occurring in this multiplier or in any of the subsequent adders will be detectable.

Double faults are detectable provided they occur in such a way as to appear at the input to the same multiplier. However at the output of the multiplier, the only detectable effect of a double fault is the same as that of a single fault. Thus this common multiplier must be the last in the network or else the double fault will be masked. As adders do not mask faults, detectable double faults can be located at points distant from the final multiplier provided the paths to that multiplier involve only adders.

The usual assumption is that a single fault is more probable than many faults. Thus the coding scheme described here is somewhat poor, in general, as it masks nearly all of the most probable faults (single faults) and most of the next most probable ones (double faults).

3. ERROR DETECTING CODES FOR MULTIPLICATION

In section 2 we have showed that the only non-separable code that commutes with both addition and multiplication is some what poor because of error masking in the multipliers. Thus it is not possible to have a single code that works well for both adders and multipliers. The AN-codes work well for addition, so it is interesting to ask what form a non-separable multiplication-preserving code must take if it is not to mask errors in this way. In this way it may be possible to either to protect a multiplier, which is a more complex circuit than an adder, against errors or construct some sort of dual code scheme for arithmetic circuits.

Although we refer to the conventional arithmetic multiplication operation throughout this paper, it should be borne in mind that there are other "multiplication" operations. It is possible [18] to think of the convolution operation as the multiplication of two polynomials. Indeed Redinbo [19] has suggested the use of non-separate error correction codes to construct a fault-tolerant FIR Filter. A BCH code is used since such codes are conveniently described using the algebra of polynomials. As BCH codes are addition preserving, Redinbo used the "one-sided" approach outlined above. In a later paper [20] he suggests an improvement which is effectively the same as using a residue (separate) code[2].

In the following we derive various necessary conditions on the form of a multiplication-preserving code. Suppose the coded form of x is c(x) and consider two values x, y. The product of the two codewords c(x), c(y) has to be the encoded form of the product of the two initial values:

$$\text{I)} \qquad c(x)c(y) = c(xy) \quad \forall\, x,y.$$

If one of the inputs was in error then ideally this ought to be detectable at the output of the multiplier i.e. the output ought not be a valid codeword. Thus another condition on the code is

$$\text{II)} \qquad c(x)v \notin C \text{ if } v \notin C, \quad \forall\, x.$$

where C is the set of codewords.

The second condition above ensures that if one of the inputs is in error then the output of the multiplier will also be in error (i.e. no error masking). The necessity of this condition may, at first, appear to be founded solely on the requirement that the code be able to check for input errors as well as for errors generated by the multiplier itself. Condition II can, however, be viewed

---

2. The correction factor that is passed between the information and check parts is in fact not necessary. Without this correction the encoding is exactly the same as a residue code modulo g(x), which is known to have the same error protection ability as the original non-separate code ([7] section 5.3).

as representing the ability to detect systematic errors within the multiplier. If a fault occurs in the control circuitry of a 'shift and add' type multiplier (say) then the effect of this fault will be the same as if the relevant input (usually designated the multiplier or coefficient) was in error. Hence if the code does not satisfy condition II, this type of error will be undetectable and the code's error-detecting ability correspondingly weak.

It is relatively easy to see that the AN code does not satisfy condition II. The next, least-complex form of encoding function to try is, perhaps, $c(x) = Ax + B$ (i.e. an AN+B code [7]).

4. AN+B CODES

We use the following notation:

if a divides b we write $a|b$,

the greatest common divisor of two integers a, b is (a,b),

the least positive residue of a modulo b is $((a))_b$.

Consider the usual setting for an arithmetic code where:

the data is taken from the ring Z_n,

and the codewords belong to the ring Z_m, where $n|m$.

Let the encoding function be $c(x) = Ax + B$,

so that $$c(x)c(y) = A^2xy + AB(x+y) + B^2$$

whereas $$c(xy) = Axy + B.$$

In order to satisfy the condition I we require that A and B satisfy the following equations:

$$A^2 = A, \qquad B^2 = B, \qquad AB = 0.$$

Clearly $A = 1$ is a possibility, but this then implies that $B = 0$ which does not lead to a useful code. This is not the only solution, however, since in Z_m there exists an idempotent element A (i.e. $A^2 = A$) whenever m is factorizable as $m = ab$, where $(a,b) = 1$ (see appendix). In this case it turns out that not only is

$$A = ((a^{-1}))_b a$$

an idempotent but also that

$$B = ((b^{-1}))_a b$$

is one as well.[3] Clearly

$$AB = ((a^{-1}))_b((b^{-1}))_a ab \equiv 0 \ MOD \ m,$$

since $ab = m$. So that provided m is factorizable in this fashion we can construct a code that satisfies condition I of section 2.

The ability of an AN+B code to detect errors, in a particular variable, is

3. NB. $B = ((1-A))_m$, see appendix.

exactly the same as that of the AN code ([7] section 4.1) i.e. it can detect any error whose residue modulo A is non-zero. Its ability not to mask errors at the input to a multiplier (condition II) can be seen as follows. Let e be an error:

$$(Ax+B + e)(Ay+B) = A(x+e)y + B + Be$$

where we have used the facts that A and B are idempotents and that AB = 0. Clearly the right hand side of the above equation will not be a codeword provided that

$$Be \not\equiv 0 \text{ MOD } m$$

or, since m = AB,

$$e \not\equiv 0 \text{ MOD } A.$$

Hence all detectable input errors (i.e. $e \not\equiv 0 \bmod A$) are still detectable at the multiplier output.

Tables 1 and 2 list some examples of idempotent AN+B codes that can be derived from the first few, known, useful AN codes. Table 1 lists the AN code generator (A'), the data ring modulus (n), the code ring modulus (m), the minimum distance (d_m) of the AN code, and hence the AN+B code, and the two idempotents (A, B). Table 2 lists a similar set of parameters for codes based on the single-error-detecting (d_m=2) AN codes with generator A'=3.

In the course of compiling tables 1 and 2 it was noted that the majority of the AN codes considered satisfied the conditions (above) for the existence of an associated idempotent AN+B code.

A′	n	m	d_m	A	B
9	7	2^6-1	2	36	28
11	93	$2^{10}-1$	2	187	837
11	3	2^5+1	3	22	12
13	315	$2^{12}-1$	2	1261	2835
13	5	2^6+1	3	26	40
19	13797	$2^{18}-1$	2	82783	179361
19	27	2^9+1	3	190	324
23	182361	$2^{22}-1$	2	729445	3464859
23	89	$2^{11}-1$	3	713	1335
29	9256395	$2^{28}-1$	2	259179061	9256395
29	565	$2^{14}+1$	3	1131	15255
37	7085	$2^{18}+1$	3	14171	247975
43	3	2^7+1	4	43	87
45	91	$2^{12}-1$	3	4005	91
47	178481	$2^{23}-1$	3	5711393	2677215
51	5	2^8-1	4	51	205
53	1266205	$2^{26}+1$	3	3798616	63310250
61	17602325	$2^{30}+1$	3	299239526	774502306
67	128207978	$2^{33}+1$	3	7179646769	1410287823
75	13981	$2^{20}-1$	3	405450	643126
87	3085465	$2^{28}-1$	3	80222091	188213365
89	23	$2^{11}-1$	4	1335	713
93	11	$2^{10}-1$	4	837	187
153	215	32895	4	7956	24940
185	1417	$2^{18}+1$	4	66600	195546
217	151	$2^{15}-1$	4	29295	3473
267	15709	$2^{22}-1$	4	2764785	1429519
315	13	$2^{12}-1$	4	2835	1261
351	23905	8390655	4	7936461	454195
353	25249	8912897	4	6564741	2348157
3937	8727391	$2^{35}-1$	3	10586325284	23773413476
6141	683	$2^{22}-1$	4	3494229	700075
6223	337	$2^{21}-1$	5	1549527	547625
13797	19	$2^{18}-1$	6	179361	82783
18631	1801	$2^{25}-1$	5	33014132	540300
25575	41	$2^{20}-1$	6	230175	818401
55831	601	$2^{25}-1$	7	7034706	26519726

69615	241	$2^{24}-1$	4	11347245	5429971
178481	47	$2^{23}-1$	8	2677215	5711393
182361	23	$2^{22}-1$	8	3464859	729445
256999	2089	$2^{29}-1$	6	237467076	299403836
486737	1103	$2^{29}-1$	7	66682969	470187943
2304167	233	$2^{29}-1$	8	232720867	304150045
2375535	113	$2^{28}-1$	6	71266050	197169406
3243933	331	$2^{30}-1$	6	363320496	710421334
9256395	29	$2^{28}-1$	10	9256395	259179061

Table 1. Codes Based on AN Codes.

n	m	A	B
5	15	6	10
85	255	171	85
341	1023	342	682
5461	16383	10923	5461
21845	65535	21846	43690
349525	1048575	699051	349525
1398101	4194303	1398102	2796202
22369621	67108863	44739243	22369621
89478485	268435455	89478486	178956970
1431655765	4294967295	2863311531	1431655765
5726623061	17179869183	5726623062	11453246122

Table 2. Single-error Detecting Codes based on A' = 3.

5. FAULT TOLERANT MULTIPLIER

Following standard coding theory [7], we define an "error" to be a non-zero difference between a variable's actual value and the correct one. The term "fault" is used to mean a physical failure which then results in an error, in one or more variables.

In order to relate error values to faults several assumptions must be made. Not only must an architecture be specified but the relative sizes of the components and faults have to be known. This is because it is, ultimately, the faults not the errors that we must guard against. Clearly a process with small enough feature size may result in a large percentage of the entire circuit being damaged by a single fault and thus produce a large error. Here we consider a relatively standard, ripple multiplier (see figure 5). Specifically, the multiplier under consideration is a parallel, asynchronous, bit-level circuit containing no broadcast lines. It is assumed that a fault will only affect a single processing element (PE). Faults in the data lines can be subsumed in to the set of faults of a PE, and thus the data lines can be considered fault free.

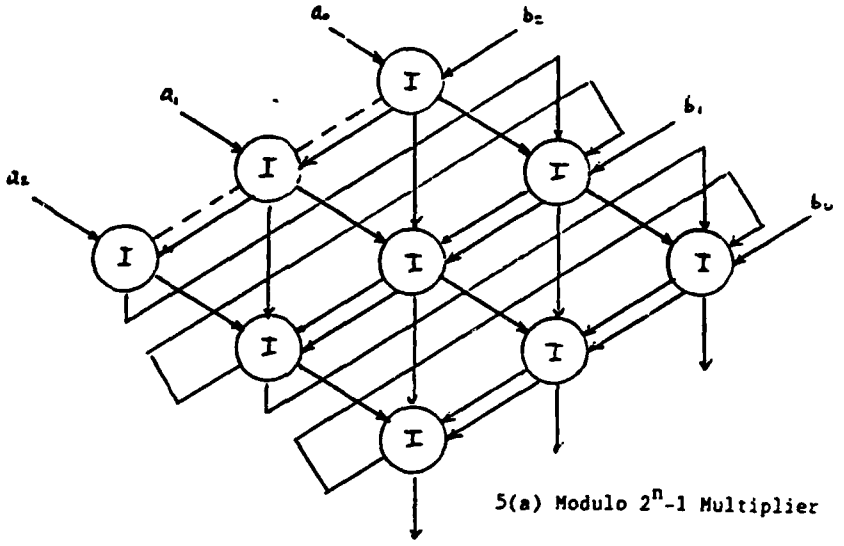
Another parameter that has to be specified is the number of faults that are expected to occur in the circuit. Here we choose to specify this value not in exact numbers but rather as a probability that a single PE will fail. Further it is assumed that the failure mode of a PE is such that some of the outputs may still function correctly. This approach allows for a complete variation from the situation of a masked fault to one where every PE output is in error.

There is only one type of PE in the chosen multiplier, consisting of a bit multiplier and input-data buffer as well as a bit-adder. Following the standard theory of arithmetic error-detecting codes [7], it is easy to see that if the adder circuitry, or associated interconnections, of any PE should fail then the multiplier output will contain an error of at most weight two. This is because any PE with a faulty adder can produce at most two erroneous outputs (the sum and carry) which will propagate undisturbed to the final output. If more than one PE fails (n say) in this manner then the output could be in error by up to a weight of 2n, subject of course to a maximum weight of half of the wordlength (see appendix, subsection 11.5).

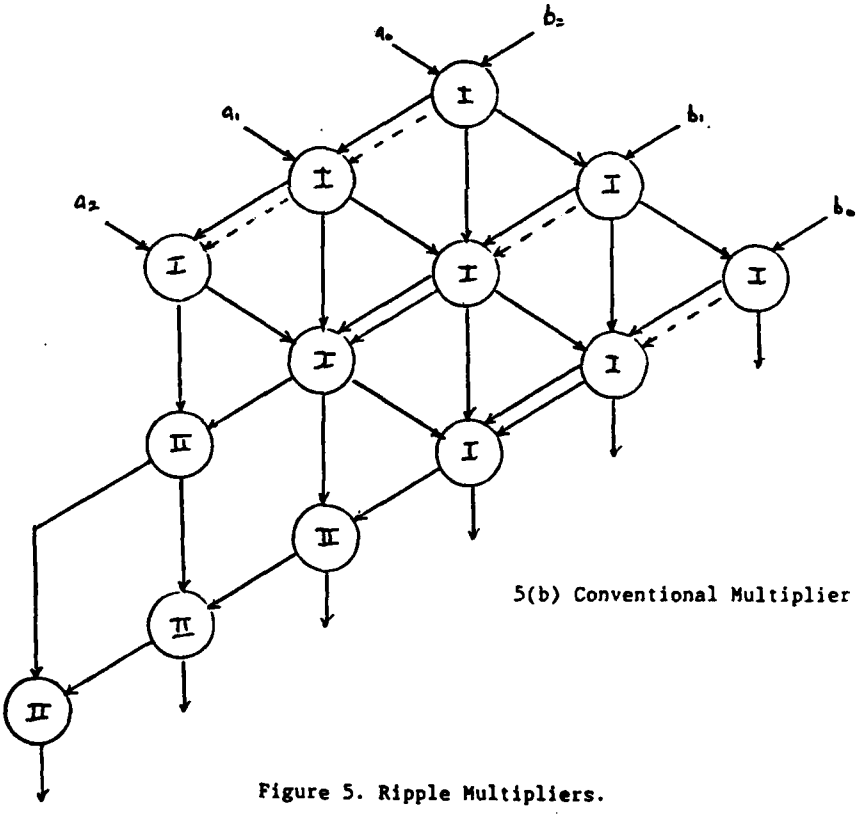
The PE's can also exhibit two other faults: the bit multiplier and the input-data buffer can fail. The former will produce a final error of weight one. The latter, however, has a more wide ranging effect. If a PE passes on, to its neighbour, an incorrect data value then this neighbour and all subsequent PE's in that chain may also produce errors (of weight one). Whether or not these errors actually occur depends upon the input data: a multiplicand bit with a (correct) value of zero clearly will mask an error in the corresponding multip-

lier bit or vice versa. Thus the weight of the error in the multiplier output is a function not only of the number of faults but also of the position of the faulty PE's and the data applied to the circuit.

At first sight, this implies that a faulty input data line in the PE in the top row/column could produce an error of weight b(b-1)/2 : assuming b PE's per row/column (and hence b bit input words). However, by design (see section 4), up to d_m-1 errors in the input data (and hence at least one) can be detected at the output. This apparent contradiction is due to the fact that the input data is coded and therefore not every bit pattern can be applied to the circuit. A simulation of the circuit is, at present, the only way to discover exactly what will happen.



5(a) Modulo 2^n-1 Multiplier



5(b) Conventional Multiplier

Figure 5. Ripple Multipliers.

6. SIMULATION RESULTS

The fault tolerant multiplier was simulated by computer program in the following manner: at each iteration random inputs were chosen, encoded in the AN+B code and applied to the multiplier routine. The latter cycled through each PE in turn calculating the various outputs. "Faulty" PE's were chosen at random, with a specified probability. The outputs (single bits) of a "faulty" PE were chosen at random in order to simulate all possible types of faults without having to specify the detailed circuitry of a PE. At the end of an iteration, the possibly erroneous result was compared to the correct value and a table of statistics updated. In one run the above procedure was repeated many times so that the variances of the estimates became acceptably small.

For the purpose of making a comparison between an AN+B coded multiplier and a more conventional fault tolerant scheme, a particular task must be considered. This is because a comparison between the scheme presented here and a DMR scheme is not straight forward. In the former case it is necessary to build a modulo 2^n-1 multiplier whereas the latter case only requires a conventional one. The problem considered here is simply that of multiplying two integers together. Figure 5 shows the sort of architecture that was considered for the conventional multiplier. The method of simulation of the DMR scheme was essentially the same as for the AN+B code scheme.

Figure 6 shows a plot of the fault detection rate for various sizes of multipliers. Table 3 list the codes used and various of their parameters: the nomenclature is the same as for table 1 with the addition of b_n and b_m which are the base 2 logarithms (i.e. equivalent number of "bits") of n and m respectively. The detection rate plotted does not include non-active faults i.e. those faults that are masked by virtue of the data values. For comparison the detection rates for a fault tolerant multipliers, of comparable size, using double modular redundancy (DMR) are also shown. Clearly the DMR schemes have consistently better detection rates, however this is not the only parameter that matters. Fault tolerance is bought by making the circuit redundant. Thus an equally important factor is the increase in circuit size.

Figure 7 is a plot of percentage increase in area of the multiplier when the codes of tables 5 and 6 are used. In calculating the increase in area, the base line was taken to be a conventional multiplier of a suitable size to handle the data range of the AN+B code. Thus both multipliers have, after any necessary decoding, the same sized output wordlength. The AN+B coded multiplier has to be a modulo 2^n-1 multiplier so its input wordlength is the same as the output wordlength. In the conventional case the input is, of course, half the size of the output.

A	B	A'	n	m	d_m	b_n	b_m	Increase (%)	
								wordlength	area
10923	5461	3	5461	16383	2	12.41	14.00	12.77	232.75
1398102	2796202	3	1398101	4194303	2	20.42	22.00	7.76	203.65
89478486	178956970	3	89478485	268435455	2	26.42	28.00	6.00	194.45
405450	643126	75	13981	1048575	3	13.77	20.00	45.23	451.27
80222091	188213365	87	3085465	268435455	3	21.56	28.00	29.89	341.31
2764785	1429519	267	15709	4194303	4	13.94	22.00	57.83	550.98

Table 3. Parameters of Simulated Codes.

The data presented in figure 7 does not take into account any of the circuitry needed to encode the data or detect an error at the multiplier output. The dashed line at 100% represents the increase in area of a DMR scheme (again ignoring the checking circuitry). The implication of these results is that, for the situation considered here, the DMR scheme is better than the AN+B code one if a dedicated fault tolerant multiplier is required. Non-intrusive real-time testing can still only be done with a "test data" type scheme such as the AN+B code one. The advantage of the AN+B code method over more conventional methods, where the test data is prepared before hand, is that generation of the test data and subsequent error detection is much easier.

It is interesting to note that the detection rate rapidly approaches 100% as the minimum distance increases. Indeed a minimum distance of 3 would appear to be quite adequate.

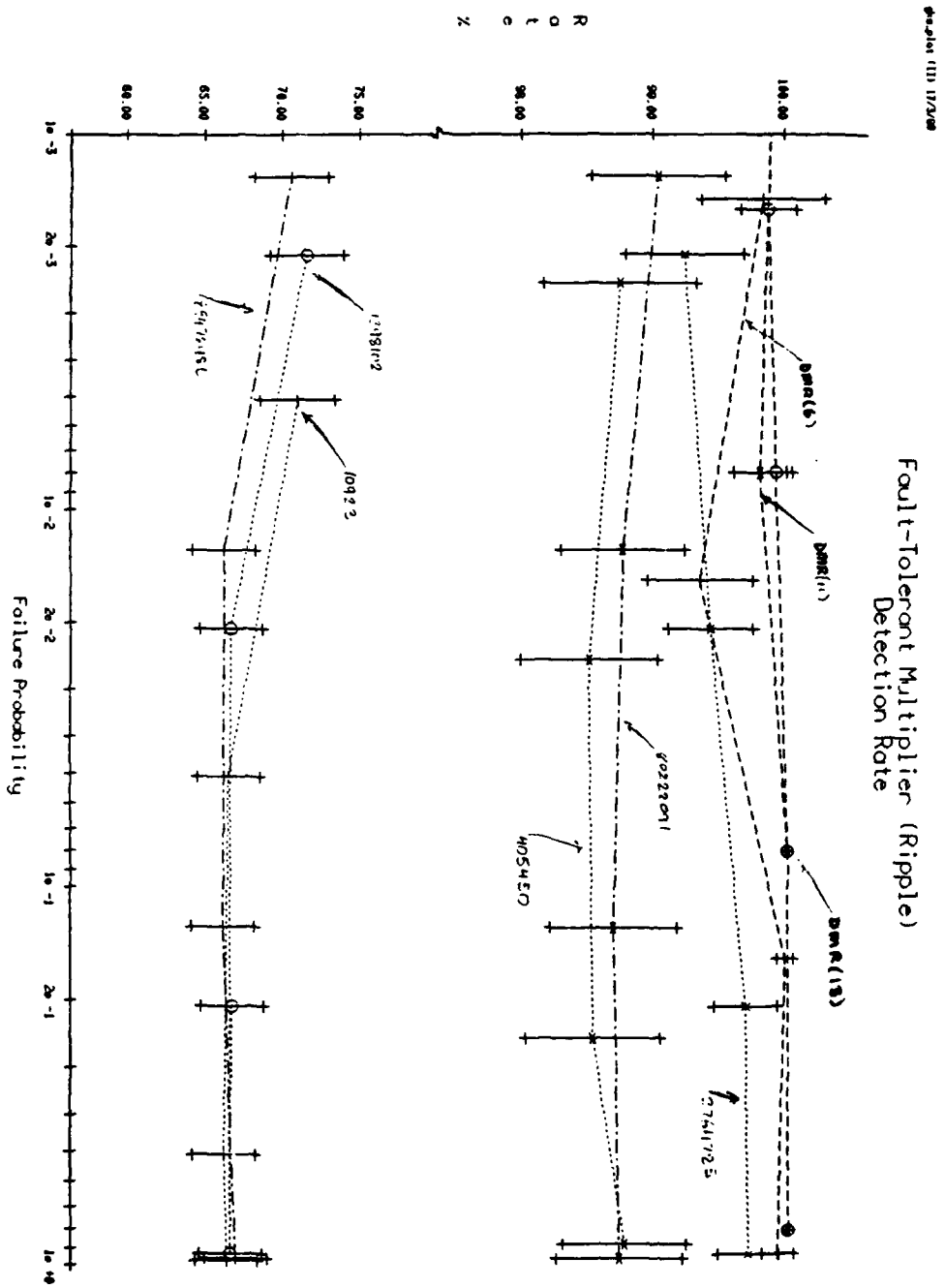


Figure 6. Detection Rate: AN+B Codes vs DMR.

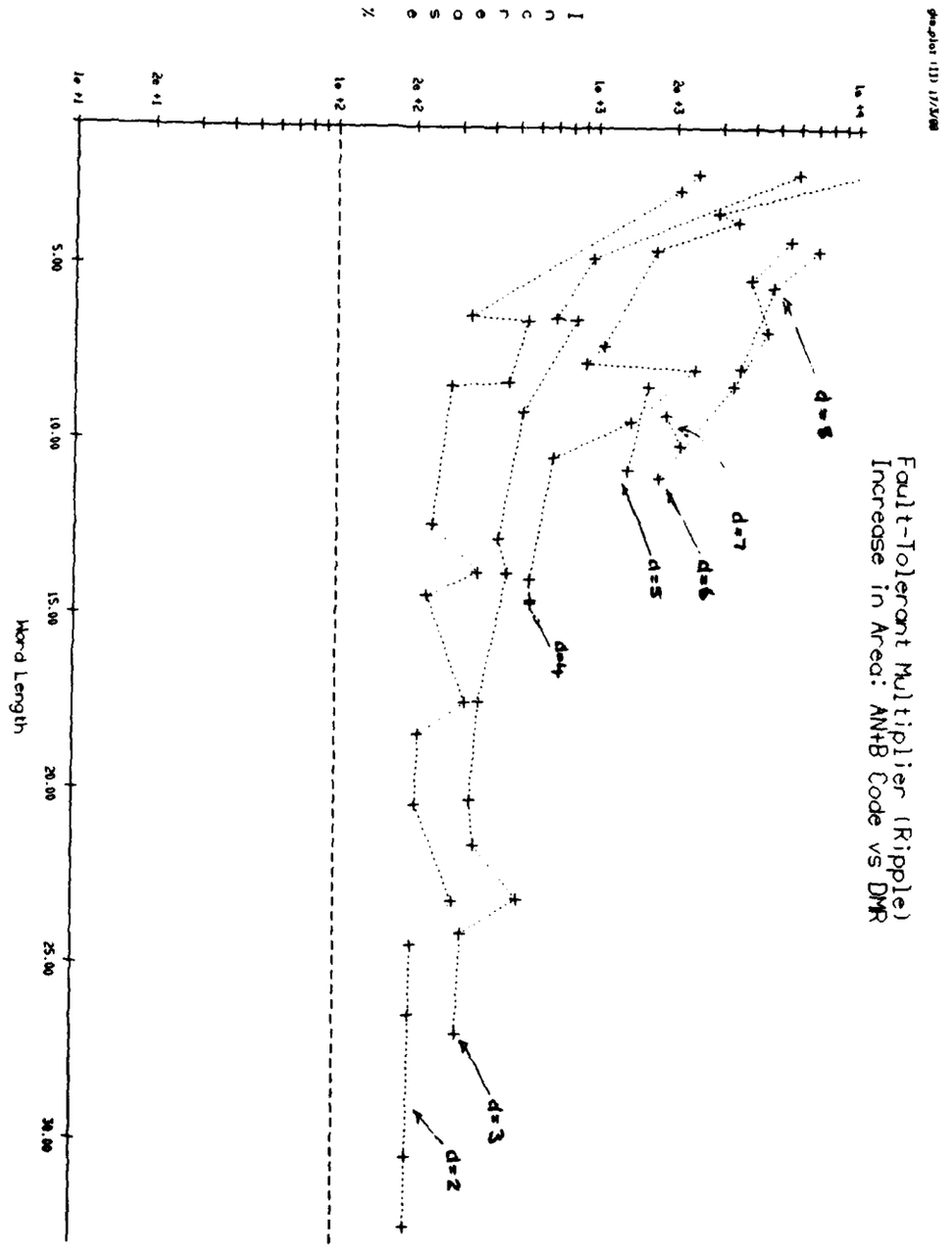


Figure 7. Increase in Area: AN+B Codes vs DMR.

7. CONCLUSION

One way to reduce the probability of a fault in the encoding/decoding circuits of an arithmetic error detection/correction code is to ensure that the arithmetic expression is more complex than the coding circuitry (algorithm fault-tolerance). In order to do this the code must preserve multiplication as well as addition. We have characterised non-separate codes that satisfy this requirement. The result is found to be consistent with the previous result about addition preserving codes.

A study of the error protection ability of this type of code shows that they are in fact rather bad at detecting errors in such arithmetic expressions. This leads us to believe that fault tolerant arithmetic, as described above, is most likely to be achieved by means of component-level fault-tolerance or the use of a separate coding scheme (residue codes).

We have also derived two conditions which an arithmetic code must satisfy in order to be used to protect a multiplier from errors. The first being the obvious statement that the code must preserve the operation of multiplication. The second requirement stems from the desire to be able to check for systematic errors and erroneous inputs.

The simplest code that satisfies the two conditions is shown to be an AN+B code where both A and B are idempotents. The necessary conditions for the existence of these idempotents was given and a list of examples produced. Such a code was shown to be able to detect all errors that are not a multiple of the idempotent A.

A specific example of a ripple multiplier was considered. It was shown that this type of code does surprisingly well in detecting errors. Indeed they can detect errors well passed their minimum distances with a high degree of success. As a result the fault tolerant multiplier based on this method could detect upwards of 95% of all active faults even if a significant number of PE's have failed. In comparison to an established technique like DMR, the use of an error-detecting code to achieve fault-tolerance was shown not to be cost-effective mainly due to a larger increase in circuit area.

8. REFERENCES

[1] Y. Savaria, N.C. Rumin, J.F. Hayes, V.K. Agraval. Soft Error Filtering: A Solution to the Reliability Problem of Future VLSI Digital Circuits. Proc. IEEE, vol.74(5), May 1986, pp669/83

[2] P. Banerjee, J. Abraham. Fault Characterisation of VLSI MOS Circuits. 1982 IEEE Conf. Circuits & Computers, pp564/8

[3] S.A. Al-Arian, D.P. Agraval. Physical Failures & Fault Models of CMOS Circuits. IEEE trans. CAS-34(3), Mar.87, pp269/79.

[4] D.K. Pradhan (Ed.), Fault Tolerant Computing: Theory & Techniques, Vols. I & II, 1986, Prentice-Hall.

[5] J.L. Diamond, Checking Codes for Digital Computers, Proc. IRE, vol 43, April 1955, pp487-488.

[6] D.T. Brown, Error Detecting and Error Correcting Binary Codes for Arithmetic Operations, IRE Trans. Electron. Comput., vol.EC-9, September 1960, pp333-337.

[7] T.R.N. Rao, Error Coding for Arithmetic Processors, 1974, Academic Press, New York.

[8] E.R. Berlekamp, Algebraic Coding Theory, 1968, McGraw-Hill, New York.

[9] F.J. MacWilliams, N.J.A. Sloane, The Theory of Error Correcting Codes, 1977, North-Holland.

[10] J. Wakerley, Error Detecting Codes Self Checking Circuits and Applications, 1978, Elsevier North-Holland.

[11] J. von Neumann, Probabilistic Logics and the Synthesis of Reliable Organisms from Unreliable Components, Automata Studies, no.34, Princeton University Press, pp43-99.

[12] K-H. Huang, J.A. Abraham, Algorithm-based Fault Tolerance for Matrix Operations, IEEE trans. Comput., vol. C-33, no.6, 1984, pp518-528.

[13] J-Y. Jou, J.A. Abraham, Fault Tolerant Matrix Operations on Multiple Processor Systems Using Weighted Checksums, SPIE Proc. vol.495, Real Time Signal Processing VII, 19-24 Aug. 1984, San Diego, CA., pp94/101.

[14] F.T. Luk. Algorithm Based Fault Tolerance for Parallel Matrix Equation Solvers. Proc SPIE vol.564, Real Time Signal Processing VIII, 1985, pp49/53.

[15] C.Y. Chein, J.A. Abraham, Fault Tolerant Systems for the Computation of Eigenvalues and Singular Values, Proc. SPIE vol.696: Advanced Algorithms and Architectures for Sig. Proc., 19-20 Aug. 1986, San Diego, Ca. pp228/37.

[16] F.T. Luk, H. Park, An Analysis of Algorithm Based Fault Tolerant Techniques, Proc. SPIE vol.696: Advanced Algorithms and Architectures for Sig. Proc., 19-20 Aug. 1986, San Diego, Ca. pp 222/7.

[17] H.S. Stone, Discrete Mathematical Structures and Their Applications, 1975, Science Research Associates, Chicago.

[18] J.H. McClellen, C.M. Rader, Number Theory in Digital Signal Processing, 1979, Prentice-Hall.

[19] G.R. Redinbo, Fault Tolerant Digital Filtering Architectures using Fast Finite Field Transforms, Signal Processing, vol.9, 1985, pp37/50.

[20] G.R. Redinbo, Protecting Convolution-type Arithmetic Array Calculations with Generalised Cyclic Codes, 8th Symp. Comput. Arith., Como, Italy, 1987, pp219-225.

[21] B. Hartley, T.O. Hawkes, Rings Modules and Linear Algebra, 1980, Chapman and Hall, London.

[22] N.S. Szabo, R.I. Tanaka, Residue Arithmetic and its Applications to Computer Technology, 1967, McGraw-Hill, New York.

[23] J.L.Massey & O.N.Garcia, Error Correcting Codes in Computer Arithmetic, chapter 5 of Advances in Information Systems Science, vol.4, J.T.Tou (Ed.), Plenum Press, 1972.

11. APPENDIX

11.1. Multiplication/addition Preserving Arithmetic Codes

A multiplication/addition preserving arithmetic code is a redundant encoding of a finite set of integers (Z_n say) such that their arithmetic structure is preserved. It is usual [7] to consider the arithmetic to be modulo the integer n ($n=2^b$ for two's complement, $n=2^b-1$ for one's complement) in which case the set Z_n can be considered to be a ring [17]. As the encoded integers are to be manipulated by a computer, or similar dedicated hardware, they too will be elements of some integer ring (Z_m, $m>n$, say). The encoding function of the code will thus be a one-to-one and into ring homomorphism (a ring monomorphism).

To prove our main result (theorem 11.1.3) we require two well known results:

Lemma 11.1.1 (See Hartley & Hawkes [21] p20.)

If ϕ: R → S is a ring homomorphism then

a) $\phi(R)=\{\phi(r)|r\in R\}$ is a subring of S.

b) let Ker ϕ = $\{r\in R|\phi(r)=0_S\}$.

ϕ is a monomorphism iff Ker ϕ = $\{0_R\}$.

[]

Theorem 11.1.2

If R is a subring of Z_n then R is a principal ideal.

Proof:

As R^+ is an additive subgroup, for $q,g\in Z_n$ we have

$$((qg))_n = ((\sum_{i=1}^{q} g))_n \in R$$

and hence R is an ideal of Z_n.

Let g be the smallest non-zero member of R. For $x\in R$ let

$$x = qg + r \qquad 0\leq r<g.$$

Then $$((r))_n = ((x - qg))_n \in R$$

since $x, ((qg))_n \in R$.

Because g is, by definition, the smallest non-zero element of R we must have that r = 0 and hence R is generated by g.

[]

With the aid of these results we may now prove

Theorem 11.1.3

If ϕ: $Z_n \rightarrow Z_m$, where $m>n$, is a ring monomorphism then $\phi(x) = Ax$ and A in an idempotent in Z_m.

Proof:

As ϕ is a ring homomorphism, by virtue of lemma 11.1.1(a) and theorem 11.1.2, $\phi(Z_n)$ is a subring of Z_m and hence a principal ideal.

Let $a \in Z_m$ be a generator of the ideal:

i.e. $\phi(Z_n) = \{ax \mid x \in Z_m\}$.

Now ϕ is actually a monomorphism hence $\phi(0) = 0$ (lemma 11.1.1(b)), thus $\phi(1) = ay$ for some non-zero $y \in Z_m$.

Then as ϕ is addition preserving we have

$$\phi(x) = \phi\left(\sum_{i=1}^{x} 1\right) = \sum_{i=1}^{x} \phi(1) = x\phi(1) = xay.$$

i.e. $\phi(x) = Ax$

where $A = ay = \phi(1)$.

But ϕ is also multiplication preserving

i.e. $\phi(x)\phi(z) = \phi(xz) \quad \forall\, x,z \in Z_n$,

or $A^2xz = Axz \quad \forall\, x,z \in Z_n$.

In particular let x=z=1, then

$$A^2 = A.$$

Hence A is an idempotent.

[]

By construction, the encoding function for an addition/multiplication preserving arithmetic codes satisfies the conditions of theorem 11.1.3. Thus we have shown that such codes have to be idempotent AN codes. This is not a totally unexpected result as it has long been known ([22] section 14.3) that a non-separate addition preserving code must be an AN code.

11.2. Idempotents

Definition:

Let R be a ring, then $e \in R$ is an idempotent if

$$e^2 = e.$$

Notation:

The (principal) ideal generated by an element j of a ring R is $\langle j \rangle$, i.e. $\langle j \rangle = \{rj \mid r \in R\}$.

In a ring R, the residue class congruent to $r \in R$, modulo an ideal J, is denoted by $[r]$, i.e. $[r] = \{s \mid s-r \in J\}$.

The ring of residue classes (or quotient ring) of a ring R modulo an ideal J is denoted by R/J.

Theorem 11.2.1

Let K be a Euclidean domain. If $j,g \in K$ are such that $g|j$ and $(g,j/g) = 1$ then, in the quotient ring $R = K/\langle j \rangle$, the ideal generated by the residue class $[g]$ can be generated by a unique idempotent.

Proof: (cf. [9] ch.8)

Define $h = j/g$. If $(h,g) = 1$ then, by the Euclidean division algorithm, there exists $p, q \in K$ such that

$$pg + qh = 1$$

and hence, in R,

$$[pg] + [qh] = [1].$$

Consider $[\gamma] = [pg] \in R$,

$$[\gamma]([\gamma] + [qh]) = [\gamma]$$

i.e.
$$[\gamma]^2 + [pg][qh] = [\gamma].$$

Now
$$[pg][qh] = [pgqh] = [pqgh] = [pqj]$$

since multiplication is commutative in a Euclidean domain.
But by definition of an ideal,

$$pqj \in \langle j \rangle$$

hence
$$[pqj] = [0].$$

So that
$$[\gamma]^2 = [\gamma]$$

and $[\gamma]$ is an idempotent.

Now as
$$[\gamma] = [p][g]$$

then
$$\langle [\gamma] \rangle \subseteq \langle [g] \rangle,$$

but
$$[g][\gamma] = [g]([1] - [qh])$$
$$= [g] - [qj]$$
$$= [g]$$

i.e.
$$\langle [g] \rangle \subseteq \langle [\gamma] \rangle$$

Thus
$$\langle [g] \rangle = \langle [\gamma] \rangle,$$

and $[\gamma]$ generates the ideal.

The idempotent $[\gamma]$ is unique, for if $[f] \in R$ is another idempotent that generates $\langle [g] \rangle$ then

$$[\gamma] \in \langle [f] \rangle$$

i.e.
$$[\gamma] = [a][f] \qquad \text{for some } [a] \in R$$

Thus
$$[\gamma][f] = [a][f]^2 = [a][f] = [\gamma].$$

Similarly $[f] \in \langle[\gamma]\rangle$

i.e. $[f] = [b][\gamma]$ for some $[b] \in R$

thus $[\gamma][f] = [b][\gamma]^2 = [b][\gamma] = [f]$.

Hence $[\gamma] = [f]$.

[]

As an example let R be the ring of integers, if $j = 65$ the ideal $\langle j \rangle$ is the set of integer multiples of 65 and the quotient ring $R/\langle j \rangle$ is the ring of integers modulo 65 (i.e. Z_{65}). Now consider the ideal of Z_{65} generated by $g = 13$ (i.e. all integers that are multiples of 13 modulo 65):

$$\langle g \rangle = \{13,26,39,52,65 \equiv 0\}.$$

We have $65/13 = 5$, and as 13 and 5 are relatively prime we can find integers p, q such that $13p + 5q = 1$.

In fact $p = 2$, $q = -5$ and hence $\gamma = 2*13 = 26$, and we have

$$\langle 26 \rangle = \{26,52,78 \equiv 13,104 \equiv 39,130 \equiv 0\} = \langle g \rangle,$$

and $(26)^2 = 676 \equiv 26 \text{ MOD } 65$.

11.3. Idempotent Generators for BCH Codes

The codewords of a BCH code form an ideal in $GF(q)[x]/\langle x^n-1\rangle$, for $(q,n) = 1$, generated by $g(x)$, a factor of x^n-1. It is well known that the roots of $g(x)$ consist of conjugate sets ([9] p199) and hence that $(g(x),(x^n-1)/g(x)) = 1$. Thus the conditions of the above theorem hold and the code can be generated by the idempotent

$$\gamma(x) = ((p(x)g(x)))_{x^n-1}$$

where $p(x) \equiv (((g(x))^{-1}))_{(x^n-1)/g(x)}$.

11.4. Idempotent Generators for AN Codes

An AN code is an ideal in Z_m, generated by A', a divisor of m. Thus provided that $(A', m/A') = 1$, the code can be generated by the idempotent

$$A = ((pA'))_m$$

where $p = ((A'^{-1}))_{m/A'}$.

The condition that A' and m/A' be relatively prime does not hold for all useful AN codes. It is, however, true for a high percentage of published values (see tables 1 and 2).

11.5. Maximum Weight n-Bit Number

It is well known [7] that the arithmetic weight of a number is the same as the Hamming distance between the binary representations of the number and three times the number. Now if x is an n-bit number then 3x is possibly an (n+2)-bit number. Now as 3 is an odd number, 3x is odd or even as x is odd or even and thus 3x and x can differ in at most n+1 places. Further more it can be shown

that 3x and x cannot differ in two consecutive places, so that the maximum weight of an n-bit number is

$$n/2 \text{ if } n \text{ is even}$$
$$(n-1)/2 + 1 \text{ if } n \text{ is odd.}$$

11.6 Binary AN Codes

A binary AN code is an arithmetic coding scheme that represents an integer N by the radix 2 representation of the integer AN, for some integer A. Although codes can be constructed for the (infinite) set of all integers, most interest is centred on finite integer rings and especially ones where the resultant arithmetic is either wrap-around carry or borrow.

The main problem for the theorist is to either a) be able to calculate the minimum distance of the code for given A; or b) calculate a value for A which gives a required minimum distance. A lot of work has been done towards these ends but as yet no succinct results are available. It is still the case that a (computer) search has to be done to get the answers, although most of the results so far obtained are useful in reducing the amount of searching required.

This section contains a list of the first few, known, AN codes along with various parameters. Also included are all pertinent theorems to date.

The table of AN code generators (table 4) includes the following parameters.:

- A: the code generator.
- M: the modulus of the information ring (i.e. $0 \leq N < M$).
- AM: the modulus of the code ring (i.e. $0 \leq AN < AM$), including (where relevant) an expression of the form $2^n \pm 1$.
- d_m: the minimum distance of the code.
- α, β: the idempotent generator α and its dual β (if any).

NB. 1. A code with A=3 can detect single errors for $M=(2^n-1)/3$ and any even n.

2. Some of the parameters for codes with $d_m=4$ are missing. The existence these codes is mentioned in Szabo & Tanaka [22] but no further details are given. There is no theorem to fill in the missing details: a computer search would be required.

3. Values for idempotent generators, if any, can only be calculated for those codes that have an explicit value for A and M.

A	M	AM		d_m	α	β
3	5	15	(2^4-1)	2	6	10
3	21	63	(2^6-1)	2		
3	85	255	(2^8-1)	2	171	85
3	341	1023	$(2^{10}-1)$	2	342	682
3	1365	4095	$(2^{12}-1)$	2		
3	5461	16383	$(2^{14}-1)$	2	10923	5461
3	21845	65535	$(2^{16}-1)$	2	21846	43690
3	87381	262143	$(2^{18}-1)$	2		
3	349525	1048575	$(2^{20}-1)$	2	699051	349525
3	1398101	4194303	$(2^{22}-1)$	2	1398102	2796202
3	5592405	16777215	$(2^{24}-1)$	2		
3	22369621	67108863	$(2^{26}-1)$	2	44739243	22369621
3	89478485	268435455	$(2^{28}-1)$	2	89478486	178956970
3	357913941	1073741823	$(2^{30}-1)$	2		
3	1431655765	4294967295	$(2^{32}-1)$	2	2863311531	1431655765
3	5726623061	17179869183	$(2^{34}-1)$	2	5726623062	11453246122
9	7	63	(2^6-1)	2	36	28
11	3	33	(2^5+1)	3	22	12
11	93	1023	$(2^{10}-1)$	2	187	837
13	5	65	(2^6+1)	3	26	40
13	315	4095	$(2^{12}-1)$	2	1261	2835
19	27	513	(2^9+1)	3	190	324
19	13797	262143	$(2^{18}-1)$	2	82783	179361
21	3	63	(2^6-1)	3		
23	89	2047	$(2^{11}-1)$	3	713	1335
23	182361	4194303	$(2^{22}-1)$	2	729445	3464859
29	565	16385	$(2^{14}+1)$	3	1131	15255
29	9256395	268435455	$(2^{28}-1)$	2	259179061	9256395
37	≈1·86E9	≈6·87E10	$(2^{36}-1)$	2	(NB:3)	
37	7085	262145	$(2^{18}+1)$	3	14171	247975
43	3	129	(2^7+1)	4	43	87
45	91	4095	$(2^{12}-1)$	3	4005	91
47	≈1·49E12	≈7·04E13	$(2^{46}-1)$	2	(NB:3)	
47	178481	8388607	$(2^{23}-1)$	3	5711393	2677215
51	5	255	(2^8-1)	4	51	205
53	≈8·50E13	≈4·50E15	$(2^{52}-1)$	2	(NB:3)	
53	1266205	67108865	$(2^{26}+1)$	3	3798616	63310250
61	≈1·89E16	≈1·15E18	$(2^{60}-1)$	2	(NB:3)	
61	17602325	1073741825	$(2^{30}+1)$	3	299239526	774502306

A	M	AM		d_m	α	β
67	128207978	8589934593	$(2^{33}+1)$	3	7179646769	1410287823
71	≈4·84E8	≈3·44E10	$(2^{35}-1)$	3	(NB:3)	
75	13981	1048575	$(2^{20}-1)$	3	405450	643126
79	≈6·96E9	≈5·50E11	$(2^{39}-1)$	3	(NB:3)	
83	≈2·65E10	≈2·20E12	$(2^{41}+1)$	3	(NB:3)	
87	3085465	268435455	$(2^{28}-1)$	3	80222091	188213365
89	23	2047	$(2^{11}-1)$	4	1335	713
93	11	1023	$(2^{10}-1)$	4	837	187
101	≈1·11E13	≈1·13E15	$(2^{50}+1)$	3	(NB:3)	
103	≈2·19E13	≈2·25E15	$(2^{51}+1)$	3	(NB:3)	
105	39	4905	$(2^{12}-1)$	4		
153	215	32895		4	7956	24940
185	1417	262145	$(2^{18}+1)$	4	66600	195546
217	151	32767	$(2^{15}-1)$	4	29295	3473
267	15709	4194303	$(2^{22}-1)$	4	2764785	1429519
315	13	4095	$(2^{12}-1)$	4	2835	1261
351	23905	8390655		4	7936461	454195
353	25249	8912897		4	6564741	2348157
357	3	1071		5		
357	46995	16777216	$(2^{24}-1)$	4		
393				4	(NB:2)	
547				4	(NB:2)	
555				4	(NB:2)	
1197	219	262143	$(2^{18}-1)$	6		
3937	8727391	34359738367	$(2^{35}-1)$	3	10586325284	23773413476
4161	63	262143	$(2^{18}-1)$	3		
6141	683	4194303	$(2^{22}-1)$	4	3494229	700075
6223	337	2097151	$(2^{21}-1)$	5	1549527	547625
13797	19	262143	$(2^{18}-1)$	6	179361	82783
18631	1801	33554431	$(2^{25}-1)$	5	33014132	540300
25575	41	1048575	$(2^{20}-1)$	6	230175	818401
55831	601	33554431	$(2^{25}-1)$	7	7034706	26519726
69615	241	16777215	$(2^{24}-1)$	4	11347245	5429971
178481	47	8388607	$(2^{23}-1)$	8	2677215	5711393
182361	23	4194303	$(2^{22}-1)$	8	3464859	729445
256999	2089	536870911	$(2^{29}-1)$	6	237467076	299403836
486737	1103	536870911	$(2^{29}-1)$	7	66682969	470187943
2304167	233	536870911	$(2^{29}-1)$	8	232720867	304150045
2375535	113	268435455	$(2^{28}-1)$	6	71266050	197169406

A	M	AM		d_m	α	β
3243933	331	1073741823	$(2^{30}-1)$	6	363320496	710421334
9256395	29	268435455	$(2^{28}-1)$	10	9256395	259179061
≈4·84E8	71	≈3·44E10	$(2^{35}-1)$	12	(NB:3)	
≈1·86E9	37	≈8·87E10	$(2^{36}-1)$	12	(NB:3)	
≈8·50E13	53	≈4·50E15	$(2^{52}-1)$	17	(NB:3)	
≈1·89E16	61	≈1·15E18	$(2^{60}-1)$	20	(NB:3)	

Table 4. The First Few Known AN Codes

11.6.1 AN Code Theorems

The proofs of the following theorems can be found in Rao[7] or Massey and Garcia[23].

7.1 For a radix r code, $d_m \geq 2$ if (A,r) = 1, and A > r.

7.2 For odd A, $M_2(A,3) = (2^k \pm 1)/A$ where[4] k is the smallest integer such that $A \mid (2^k \pm 1)$.

7.3 If (A,r) = 1 then $k = e_r(A)$ or[5] $e_r(A)/2$ as $e_r(A)$ is odd or even respectively.

7.4 If A is an odd prime and 2 is primitive in GF(A)
then $M_2(A,3) = (2^{(A-1)/2}+1)/A$.
If A is an odd prime and 2 is not primitive in GF(A)
then $M_2(A,3) = (2^{(A-1)/2}-1)/A$.

7.5 Let $A = p_1 p_2$ be the product of two distinct primes. If $e_i = e_2(p_i)$ i=1,2 and $L = \langle e_1, e_2 \rangle$ then

$M_2(A,3) = (2^{L/2}+1)$ if (e_1,e_2) is even and both e_1/d and e_2/d are odd,
$= (2^L-1)$ otherwise.

7.6 (BM Codes) If B is a prime and 2 is primitive in GF(B) then
$A = (2^{(B-1)}-1) \Rightarrow d_m = \lfloor (B+1)/3 \rfloor$.
If -2, but not 2, is primitive in GF(B) then
$A = (2^{(B-1)/2}-1) \Rightarrow d_m = \lfloor (B+1)/6 \rfloor$,
but then $A = (2^{(B-1)}-1)$ is a repetition code with $d_m = 2\lfloor (B+1)/6 \rfloor$.

23.1 If A (odd) > 1 then $d_m \geq 3$ iff either a) $e_2(A)$ is odd or b) $e_2(A)$ is even but $A \nmid (2^{e/2}+1)$.

23.2 If A = 3 and $M = (2^n-1)/3$ for n even, then $d_m = 2$.

---

4. $M_r(A,d)$ is the smallest integer such that the number $A*M_r(A,d)$, in radix r, has weight < d, hence the radix-r code with generator A has minimum distance d provided the data is taken from Z_M.
5. $e_r(A)$ is the exponent of r modulo A i.e. $r^e \equiv 1$ MOD A.

DOCUMENT CONTROL SHEET

Overall security classification of sheet ..... UNCLASSIFIED .....

(As far as possible this sheet should contain only unclassified information. If it is necessary to enter classified information, the box concerned must be marked to indicate the classification eg (R) (C) or (S))

1. DRIC Reference (if known)	2. Originator's Reference	3. Agency Reference	4. Report Security Classification
	Memorandum 4215		Unclassified

5. Originator's Code (if known)	6. Originator (Corporate Author) Name and Location
7784000	Royal Signals and Radar Establishment St Andrews Road, Malvern, Worcestershire WR14 3PS

5a. Sponsoring Agency's Code (if known)	6a. Sponsoring Agency (Contract Authority) Name and Location

7. Title

NON-SEPARATE ARITHMETIC CODES

7a. Title in Foreign Language (in the case of translations)

7b. Presented at (for conference papers) Title, place and date of conference

8. Author 1 Surname, initials	9(a) Author 2	9(b) Authors 3,4...	10. Date	pp. ref.
Proudler I K			9.88	30

11. Contract Number	12. Period	13. Project	14. Other Reference

15. Distribution statement

Unlimited

Descriptors (or keywords)

continue on separate piece of paper

Abstract

It is shown that a non-separate arithmetic code that preserves both addition and multiplication must be an AN code where the generator A is an idempotent element of the ring being used. An idempotent element is one that satisfies the equation $x^2 = x$. Given this type of code, its ability to detect errors in arithmetic expressions is explored and shown to be poor, due to error masking in multipliers.

The constraints placed on a non-separate multiplication-preserving arithmetic code that avoids such problems are discussed. The simplest code satisfying these conditions turns out to be an AN+B code where both A and B are idempotent elements. Conditions for the existence of this type of code are given along with a list of examples. The fault tolerance provided by these codes is then considered for a [specific example.